# An encryption algorithm based on catalogued shuffled keyboards

Andrew Garcia, PhD

**Abstract**

Good encryption can be used to protect data and private information. When properly encrypted, even if data is accessed in an unauthorized manner or unwillingly disclosed, the non-consented reader will be unable to read it without the correct encryption keys. The algorithm presented here, **streamdice**, is a stream cipher which encrypts characters (i.e. letters, numbers and some allowed signs) by both their specific identity as well as their relative location in the message thread. For streamdice, the stream units are shuffled keyboards generated by a pseudo-random number generator (PRNG), each of which are shifted once for every single encrypted character. The shuffled keyboards are limited and kept in memory with hashes, which are in turn dependent on the provided keys for encryption. The pseudo-random factor obfuscates the periodicity of the algorithm, and the encryption operations make it challenging to exploit by brute force.

## 1   Method

The cipher takes a message and splits it into separate characters.

This method uses a hashmap where each one of its keys corresponds to an index of the keyboard character representing it in QWERTY order. For instance, the first keys for Q, W, and E characters are 1, 2 and 3, respectively. This ordered arrangement is analogous to a symbolic keyboard as the one in Figure 1.

For the encryption, the character values of the hashmap are shuffled with a pseudo-random number generator (PRNG) seeded by a "hash" number. For instance, shuffling the original keyboard in Figure 1 with a Mulberry32 PRNG and a #5443 hash will give the shuffled keyboard in Figure 2. Because the values are shuffled, the keys will be preserved, and thus the ciphertext can be transformed back by reference to the original keyboard, i.e. $W(\text{ciphertext}) \rightarrow 4(\text{index}) \rightarrow R(\text{text})$. The keyboard is shuffled each time a new character is encrypted with N permutations, calling a new shuffling operation by its PRNG hash as shown in Figure 3. As seen in the same Figure, the permuted keyboards repeat periodically if the number of keyboards is less than the number of characters to encrypt. The specific hashes are computed directly from the 2 keys provided by the user for the encryption. An **H** vector contains all the $H_i$ hashes used to generate the shuffled keyboards. The $N$ number of $H_i$ hashes is equal to the number of digits provided for $\text{key}_2$ and are computed in the following way:

$$\Gamma_i = (\text{key}_2 // 10^i)\%10 \tag{1}$$

$$H_i = \text{key}_1 + \Gamma_i \tag{2}$$

The hashes used to generate the new keyboard, rather than the specific keyboard arrangement, are the objects kept in memory thorughout the encryption. As suggested above, the decryption takes the keys used to encrypt the messages and reverses the protocol. This method, thus, optimizes auxiliary memory rather than encryption time complexity.
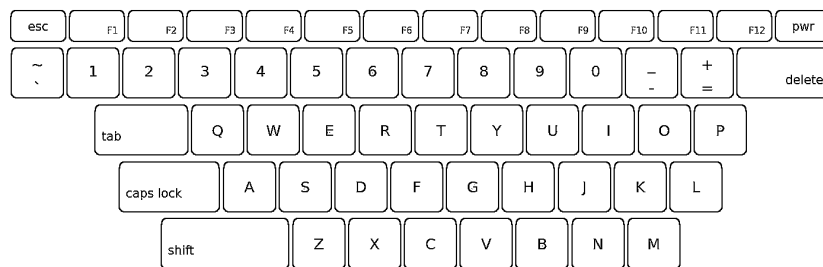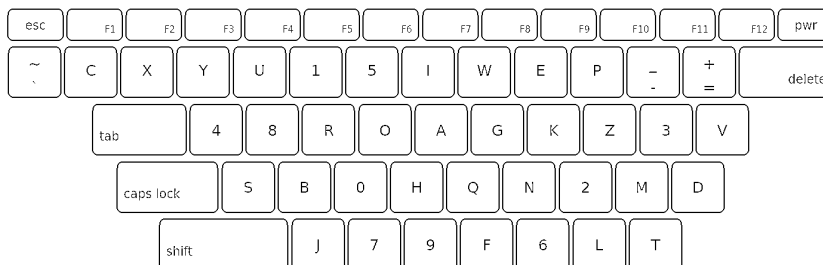
Figure 1: Standard QWERTY keyboard



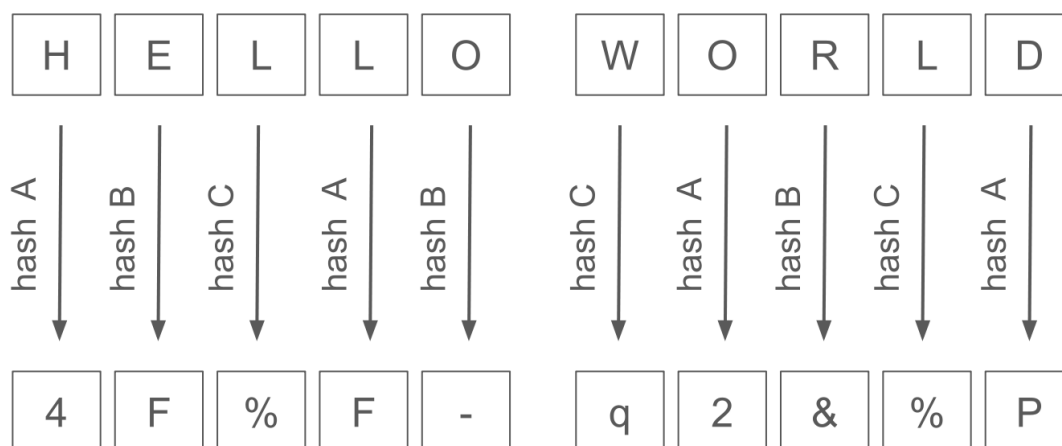Figure 2: Randomly-shuffled keyboard with #5443



Figure 3: Encrypting "hello world" with a periodically-repeating stream of 3 shuffled keyboards.